Article
Eduard Lebedyuk · Aug 7, 2020 5m read

Open Exchange

Containerising .Net/Java Gateways (or Kafka Integration Demo)

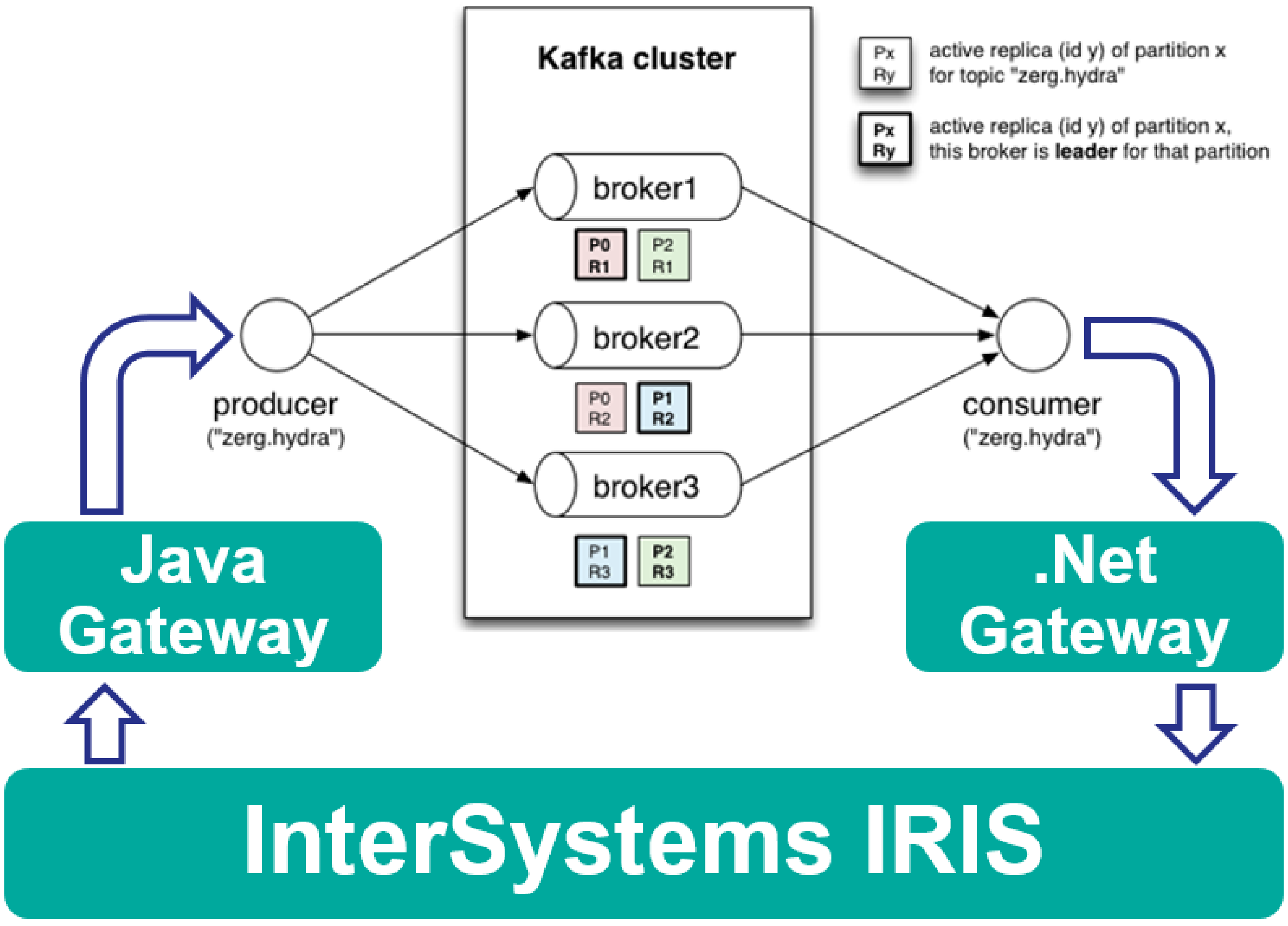
In this article, I will show how you can easily containerize .Net/Java Gateways.

For our example, we will develop an Integration with Apache Kafka.

And to interoperate with Java/.Net code we will use PEX .

Architecture

Our solution will run completely in docker and look like this:



Java Gateway

First of all, let's develop Java Operation to send messages into Kafka. The code can be written in your IDE of choice and it can look like this.

In short:

- To develop new PEX Business Operation we need to implement abstract `com.intersystems.enslib.pex.BusinessOperation` class
- Public properties are Business Host Settings
- `OnInit` method is used to init connection to Kafka and get a pointer to InterSystems IRIS
- `OnTearDown` is used to disconnect from Kafka (at process shutdown)
- `OnMessage` receives dc.KafkaRequest message and sends it to Kafka

Now let's pack it into Docker!

Here's our dockerfile:

```
FROM eclipse-temurin:8-jre-alpine AS builder

ARG APP_HOME=/tmp/app

COPY src $APP_HOME/src

COPY --from=intersystemscommunity/jgw:latest /jgw/*.jar $APP_HOME/jgw/

WORKDIR $APP_HOME/jar/
ADD https://repo1.maven.org/maven2/org/apache/kafka/kafka-clients/2.5.0/kafka-
clients-2.5.0.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-classic/1.2.3/logback-
classic-1.2.3.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-core/1.2.3/logback-
core-1.2.3.jar .
ADD https://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.7.30/slf4j-api-1.7.30.jar .

WORKDIR $APP_HOME/src

RUN javac -classpath $APP_HOME/jar/*:$APP_HOME/jgw/* dc/rmq/KafkaOperation.java && \
    jar -cvf $APP_HOME/jar/KafkaOperation.jar dc/rmq/KafkaOperation.class

FROM intersystemscommunity/jgw:latest

COPY --from=builder /tmp/app/jar/*.jar $GWDIR/
```

Let's go line by line and see what's going on here (I assume familiarity with multi-stage docker builds):

```
FROM eclipse-temurin:8-jre-alpine AS builder
```

Our starting image is JDK 8 (Note that openjdk:8 image is deprecated, use one of the suggested alternatives).

```
ARG APP_HOME=/tmp/app
COPY src $APP_HOME/src
```

We're copying our sources from `/src` folder into `/tmp/app` folder.

COPY --from=intersystemscommunity/jgw:latest /jgw/*.jar $APPHOME/jgw/

We're copying Java gateway sources into `/tmp/app/jgw` folder.

```
WORKDIR $APP_HOME/jar/
ADD https://repo1.maven.org/maven2/org/apache/kafka/kafka-clients/2.5.0/kafka-
clients-2.5.0.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-classic/1.2.3/logback-
classic-1.2.3.jar .
ADD https://repo1.maven.org/maven2/ch/qos/logback/logback-core/1.2.3/logback-
core-1.2.3.jar .
ADD https://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.7.30/slf4j-api-1.7.30.jar .

WORKDIR $APP_HOME/src

RUN javac -classpath $APP_HOME/jar/*:$APP_HOME/jgw/* dc/rmq/KafkaOperation.java && \
    jar -cvf $APP_HOME/jar/KafkaOperation.jar dc/rmq/KafkaOperation.class
```

Now all dependencies are added and `javac/jar` is called to compile the jar file. For a real-life projects it's better to use maven or gradle.

```
FROM intersystemscommunity/jgw:latest

COPY --from=builder /tmp/app/jar/*.jar $GWDIR/
```

And finally, the jars are copied into base jgw image (base image also takes care of starting the gateway and related tasks).

.Net Gateway

Next is .Net Service which will receive messages from Kafka. The code can be written in your IDE of choice and it can look like this.

In short:

- To develop new PEX Business Service we need to implement abstract `InterSystems.EnsLib.PEX.BusinessService` class
- Public properties are Business Host Settings
- `OnInit` method is used to init connection to Kafka and subscribe to topics and get a pointer to InterSystems IRIS
- `OnTearDown` is used to disconnect from Kafka (at process shutdown)
- `OnMessage` consumes messages from Kafka and sends `Ens.StringContainer` messages to other Interoperability hosts

Now let's pack it into Docker!

Here's our dockerfile:

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build

ENV ISC_PACKAGE_INSTALLDIR /usr/irissys
ENV GWLIBDIR lib
ENV ISC_LIBDIR ${ISC_PACKAGE_INSTALLDIR}/dev/dotnet/bin/Core21
```

```
WORKDIR /source
COPY --from=store/intersystems/iris-
community:2020.2.0.211.0 $ISC_LIBDIR/*.nupkg $GWLIBDIR/

# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# copy and publish app and libraries
COPY . .
RUN dotnet publish -c release -o /app

# final stage/image
FROM mcr.microsoft.com/dotnet/core/runtime:2.1
WORKDIR /app
COPY --from=build /app ./

# Configs to start the Gateway Server
RUN cp KafkaConsumer.runtimeconfig.json IRISGatewayCore21.runtimeconfig.json && \
    cp KafkaConsumer.deps.json IRISGatewayCore21.deps.json

ENV PORT 55556

CMD dotnet IRISGatewayCore21.dll $PORT 0.0.0.0
```

Let's go line by line:

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build
```

We use full .Net Core 2.1 SDK to build our app.

```
ENV ISC_PACKAGE_INSTALLDIR /usr/irissys
ENV GWLIBDIR lib
ENV ISC_LIBDIR ${ISC_PACKAGE_INSTALLDIR}/dev/dotnet/bin/Core21

WORKDIR /source
COPY --from=store/intersystems/iris-
community:2020.2.0.211.0 $ISC_LIBDIR/*.nupkg $GWLIBDIR/
```

Copy .Net Gateway NuGets from official InterSystems Docker image into our builder image

```
# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# copy and publish app and libraries
COPY . .
RUN dotnet publish -c release -o /app
```

Build our library.

```
# final stage/image
FROM mcr.microsoft.com/dotnet/core/runtime:2.1
WORKDIR /app
COPY --from=build /app ./
```

Copy library dlls into the final container we will actually run.

```
# Configs to start the Gateway Server
RUN cp KafkaConsumer.runtimeconfig.json IRISGatewayCore21.runtimeconfig.json && \
    cp KafkaConsumer.deps.json IRISGatewayCore21.deps.json
```

Currently, .Net Gateway must load all dependencies on startup, so we make it aware of all possible dependencies.

```
ENV PORT 55556

CMD dotnet IRISGatewayCore21.dll $PORT 0.0.0.0
```

Start gateway on port 55556 listening on all interfaces.

And we're done!

Here's a complete docker-compose to get it all running together (including Kafka and Kafka UI to see the messages).

To run the demo you need:

1. Install:
   - docker
   - docker-compose
   - git
2. Execute:

```
git clone https://github.com/intersystems-community/pex-demo.git
cd pex-demo
docker-compose pull
docker-compose up -d
```

Important notice: Java Gateway and .Net Gateway libraries MUST come from the same version as InterSystems IRIS client.

#.NET #Best Practices #Business Operation #Business Service #Docker #Interoperability #Java #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/containerising-netjava-gateways-or-kafka-integration-demo